DOCUMENT RESUME

ED 261 661 IR 011 810

AUTHOR Ng, K. L. Evelyn
TITLE Introductory LOGO Programming for Language Teachers.
PUB DATE [84]
NOTE 37p.
PUB TYPE Guides - Classroom Use - Materials (For Learner) (051) -- Viewpoints (120)

EDRS PRICE MF01/PC02 Plus Postage.
DESCRIPTORS *Authoring Aids (Programing); Computer Literacy; *Computer Software; Elementary Secondary Education; English Teacher Education; *Language Arts; Language Proficiency; *Language Skills; Language Teachers; Learning Modules; *Microcomputers; *Programing; Programing Languages; Teacher Developed Materials
IDENTIFIERS *LOGO Programing Language

ABSTRACT

Intended for language teachers who wish to learn how to program a computer to develop instructional materials, this paper specifically recommends the LOGO programming language for their needs. The core of the article consists of program examples and suggested activities in acquiring basic LOGO programming concepts, but it also includes a rationale for the selection of LOGO as the programming language for language teachers, a discussion of programming issues related to language teachers with specific reference to LOGO, and suggested LOGO programming concepts and learning activities. Eight instructional units present objectives and activities on the following topics: (1) a general understanding of LOGO operation and basic drawing commands; (2) understanding of basic list processing commands through reading of explanations and examples; (3) writing programs using the commands PRINT, SENTENCE, and REQUEST to produce completion exercises; (4) writing programs using the commands REPEAT and MAKE, and the utility procedure SHOWPICTURE; (5) writing programs using utility procedures PICKRANDOM and READNUMBER together with command OUTPUT to generate sentences; (6) writing programs for vocabulary or reading comprehension exercises using the commands GO and IF...THEN...STOP and ELSE; (7) writing programs using the conditional commands listed in unit six and ANYOF/ALLOF to produce exercises on grammar or language usage; and (8) writing programs using recursion and the commands FIRST, LAST, BUTFIRST, and BUTLAST to produce exercises that can be used to look for key words in students' writing. A 15-item reference list completes the document. (JB)

**********
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
**********

U.S. DEPARTMENT OF EDUCATION
NATIONAL INSTITUTE OF EDUCATION
EDUCATIONAL RESOURCES INFORMATION
CENTER (ERIC)

This document has been reproduced as received from the person or organization originating it.

Minor changes have been made to improve reproduction quality.

• Points of view or opinions stated in this document do not necessarily represent official NIE position or policy.

ED261661

# Introductory LOGO Programming for Language Teachers

K. L. Evelyn Ng

Department of Measurement, Evaluation and Computer Applications,

The Ontario Institute for Studies in Education, Toronto, Ontario, Canada

IR011810

"PERMISSION TO REPRODUCE THIS MATERIAL HAS BEEN GRANTED BY

K.L. Evelyn Ng

TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)."

Introductory LOGO Programming for Language Teachers

## Introduction

That computers have penetrated into man's environment is not to be disputed. In the school, the growing prevalence of the microcomputer has freed computer use from the restricted reliance on large-scale computer-assisted instruction. It is no longer realistic for language teachers to blindly or phobically label the use of computers on only Maths, Physics, Chemistry or Business and Economics.

Undeniably, as Marty (1981) has remarked, computers represent a level of technological complexity far greater than the usual devices used in the language classroom, such as film projectors, tape and video recorders. However, their very complexity makes it a challenge for the conscientious language teachers to start learning about this pervasive instrument to improve language instruction.

This paper is intended for language teachers interested in finding out how they may build up a confident and yet practical relationship with the computer by learning to program it. The paper is a sharing of an experience in a programming venture in LOGO. While the core of the article consists of program examples, and suggested activities in acquiring basic LOGO programming concepts, the scope covers the sections listed below:

Rationale for the selection of LOGO as the programming language for language teachers.

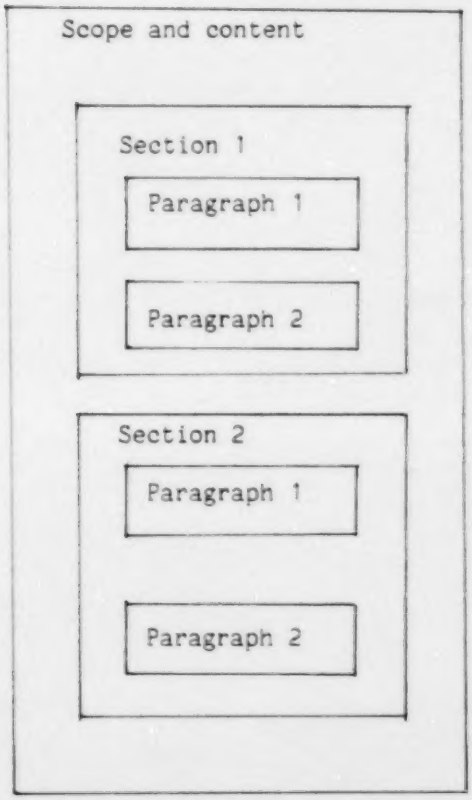
Programming issues related to language teachers with specific reference to LOGO experience.

Suggested LOGO programming concepts and learning activities.

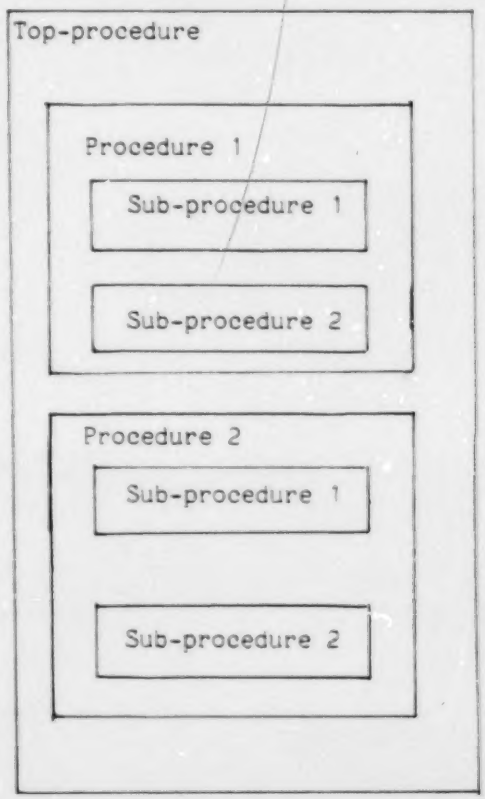
I. Rationale for the selection of LOGO as the programming language for language teachers

LOGO possesses a number of characteristics, which are appealing to the language teachers starting to learn programming. In the first place, LOGO programming emphasizes a modular structure; its hierarchical organization demands the shaping of programming instructions into levels and sub-levels. As in essay or report writing, where ideas are grouped into sections and paragraphs, instructions in LOGO programs are arranged into modules of top-procedures, procedures and sub-procedures. This analogy between essay structure and LOGO program structure can be illustrated in the diagrams below:

Essay structure




LOGO program structure



However, there is one big distinction in the analogy: in computer programs, the sub-procedures used in one procedure, say Procedure 1, can be the same sub-procedures employed in another procedure, for example, Procedure 2. In essay writing, nevertheless, paragraphs written in one section are rarely, or never, used again in another section. In this respect, program writing is easier than essay writing.

Secondly, the list processing features of LOGO make it highly commendable for text manipulation. In the language, the list is the main information or data grouping mechanism. Lists have no fixed size, and do not have to be pre-specified. In many languages, such as BASIC and FORTRAN, the grouping mechanism is the array, and the size or dimension of the array has to be declared at the beginning of the program. The very absence of such terms as arrays and dimensions is welcoming to language teachers, who, in general, are not mathematically-oriented. Because LOGO lists are not fixed in size, there is greater freedom or flexibility in the process of instructing the computer what to do. There are no problems caused by data description and input/output formatting difficulties. Besides, LOGO lists can contain different types of data: numbers, words or other lists. Variables do not have to be declared as integers, or characters. The object of a list can be a character, a word, a phrase or a sentence. This parallel with the elementary objects in the English language renders LOGO almost ideal for language arts programming.

Thirdly, to help correcting mistakes in a program, LOGO offers error messages that are more meaningful to the computer layman. Instead of giving cold and mechanical warnings like: "Operators messages not allowed" and "Invalid character in statement", LOGO provides feedback such as:

[Something] needs more inputs
[Something] does not like [another thing] as input.

Moreover, the interactive nature of the language enables the programmer to go to where the mistake occurs, and to type in commands to find out what is happening at that point, thus eliminating possible error-causing factors, and ultimately locating the mistake (Harvey, 1982).

Fourthly, LOGO is at present available in different versions on practically all major kinds of microcomputers: Terrapin/Krell LOGO for the Apple II, TI LOGO for Texas Instruments 99/4A, Commodore LOGO for Commodore 64, IBM and DR LOGO for the IBM microcomputers, and EXPERLOGO and MACLOGO for the Macintosh, etc. More importantly, various versions are continually being upgraded for larger memory capacity, and greater speed. For instance, Terrapin has announced the introduction of a new version, Terrapin LOGO 3.0, for the Apple family, and the release of a LOGO cartridge for the Commodore Plus/4 (The National LOGO Exchange, 1985). The 3.0 version allows the Apple user to take advantage of the full memory capabilities of the computer in use, 64K or 128K, and the new LOGO cartridge makes available roughly twice the memory space of Commodore LOGO. The wide availability of LOGO on different types of microcomputer means that practice and implementation on the part of the teacher are less restricted.

Fifthly, the graphics capability of the language is great; graphics programming is possible even for beginners. In LOGO, the use of the turtle to draw sketches has been designed with the children population in mind. By asking the turtle to move forward or backward, or to turn a certain degree to the left or right, diagrams can be drawn without much programming difficulty. Additionally, some packages on LOGO have been developed with easy-draw programs. For example, Daniel Watt has produced some such programs on disks that accompany his book, Learning with LOGO (1983a). In many other programming languages, such as BASIC, the complexity of graphics programming is enormous, because most approaches to computer graphics are based on Cartesian coordinates. Each line drawn is specified in terms of the positions of the endpoints, relative to a fixed X, Y coordinate system. Thus, before the positions of the endpoints can be specified, coordinates must be calculated first, very often having to use trigonometry (Harvey). But in LOGO, to draw a box for instance, one only has to ask the turtle to repeat four times, moving forward, say 50 steps, then turning right 90 degrees. For the language teacher, this capability of allowing for relatively easy production of drawings and sketches is a big help, if not a necessity.

Lastly, LOGO is a language in which beginners can start writing and running programs almost right away. Provided one can type accurately, the list processing capability enables the novice to produce sensible programs even when only a few commands have been introduced; there is not a large amount of detailed information that he/she has to remember (Du Boulay & O'Shea, 1981). Furthermore, as Howard Austin (1976) has said in his report, "Teaching Teachers LOGO - the Lesley experiments", 'one of the best things about learning something new is the fact that

you can then teach someone else what you just learned (p. 23).' LOGO learners share such enthusiasm and confidence-inspiring spirit, for LOGO is relatively new as a microcomputer programming language and learning environment.

II. Programming issues related to language teachers with specific reference to LOGO experience

Before embarking on any programming activities, it may be helpful if language teachers pay some heed to a number of related issues.

Fundamental programming attitude

As Papert (1980) says in Mindstorms, 'When you learn to program a computer you almost never get it right the first time (p. 23).' To write programs that do not run the first or second time is almost a rule. This does not mean that teachers should not try to get programs correct when they plan and write them, but they should adopt, when beginning to learn programming, the attitude of not being afraid of making mistakes, and more positively, the spirit of facing mistakes, finding them out and correcting them. (In my first experience with programming in Fortran, because of the large number of people attending the course and for the shortage of available facilities, actual computer time was very short. To complete the assignments, students barely had time to type in the program. I dreaded making mistakes; I could not afford the time to have the programming exercise wrong. Thus, when a mistake was made, even if it was a simple syntactic one, I did not try to figure out from the error message what the problem was. Assistance was sought from the tutor right away. It never occurred to me then that programming skills could be acquired without a tutor around.) As locating and eliminating errors are integral tasks of programming, two working habits are of vital advantage. The first is good planning, and the second is clear organization. In the absence of these, tracing the whole program to spot errors, and error-causing factors is difficult and

time-consuming, unless the mistakes made are only superficial typing or syntactic ones.

Major conceptual distinction between computer instruction and human instruction

No doubt, the Peanut Butter and Jelly Sandwiches example, suggested by Marilyn Burns (1980) demonstrates very clearly how even children can be introduced to the skill of expressing ideas precisely, and in a logical order based on everyday life experience. But from the list of instructions given to a child on how to prepare the sandwiches, one will find a number of difficulties if they are applied to the computer. For instance, given the instruction, "Get 2 slices of bread" (p. 29), a computer would have problem with 'slices': What is a slice? Is it the same as bread? How big? Again, a computer would query the instruction "Put the top back on the peanut butter jar" (p. 29) with questions like: Which jar? How about if the top does not fit the jar? In human instruction, a lot of messages to be put across are taken for granted, understood and inferred. Du Boulay and O'Shea have rightly remarked that 'teachers often say that writing a computer program is just like giving a person instructions, but it is not...the specification for human use [is] often "qualification" rather than "conditional" [as in the difference between] "Put red things in Box 1" and "If thing is red then put in Box 1" (p. 149).' A grasp of this conceptual distinction is essential to the forming of appropriate logic -- instruction flow -- in program writing, and to the understanding of present computer limitations.

Levels of programming

As computer literacy can be graded into levels, programming literacy can also be identified into different stages. The language teachers who start learning programming should know what to aim at, and where to go to at the completion of a stage. There are many ways of identifying these levels. According to Pea and Kurland (1984) who have had programming experience themselves, and who have taught adults and children to program in LOGO, four levels can be identified. The second and third levels specified may give teachers some ideas as to what to expect of themselves at the beginning and subsequent stages. (Level I is simply the program user, who has learnt to execute already written programs such as games or CAI. Level IV, identified as the software developer, is approaching professional programming.)

> Level II: Code Generator. At this level the student knows the syntax and semantics of the more common commands in a language. He or she can read someone else's program and explain what each line accomplishes. The student can locate "bugs" [mistakes] preventing commands from being executed (e.g. syntax errors), can load and save program files to and from an external storage device, and can write simple programs of the type he or she has seen previously... (p. 152)

> Level III: Program Generator. At this level the student has mastered the basic commands ... [and] can read a program and explain its purpose, what functions different parts of the program serve, and how the different parts are linked together. The student can locate bugs that cause the program to fail to function

> properly or bugs that cause the program to crash [break-down] as a result of unanticipated conditions or inputs. The student can load, save, and merge files and can do simple calls to and from files from inside the main program... (p. 153)

Particular programming needs of language teachers

What kinds of programming activities would language teachers like to be engaged in? What are their particular needs and requirements? According to the Interim Report of the OISE Survey and Analysis of Microcomputer Software for Language Arts (1983), the two most frequently used types of software are drills to reinforce previously taught knowledge and skills, and word processing (p. 8). Again, composition and drills are the two options at the top of the list of purposes the microcomputers can best serve in teaching language arts. The specific areas of the language arts curriculum for which teachers think software is most needed are reading comprehension, language use situations and grammar (p. 9). These data, in addition to what individual teachers consider essential, may be taken as a framework within which teachers practise their programming skills. For example, they may find greater incentive in starting with commands and operations which enable them to produce some useful drill and writing exercises. Editing operations for a certain amount of word processing are not to be overlooked, and some commands for the production of simple drawings may also be helpful, as graphics is vital in the development of appealing language exercises.

Common LOGO difficulties

In most studies conducted to investigate how beginners learn LOGO, the subjects involved have been mainly children. Although it is suggested that adults make the same kind of mistakes as children when both are programming novices (Austin, 1976; Statz, 1973), the difficulties reported in these studies should perhaps be viewed with some reservation if adults are involved. However, from the accounts of children's experience with LOGO, teachers may anticipate some possible pitfalls, and thus try to avoid them. Examples of difficulties recorded include the failure to

-differentiate between defining and executing a procedure,

-grasp the distinction between the name and value of a variable,

-break-down problems into units, allowing defined procedures to be used as building blocks,

-use and terminate a recursion, and

-use conditional tests (Du Boulay & O'Shea).

## III. Suggested LOGO programming concepts and learning activities

As teachers are used to working according to guidelines, although LOGO designers favour and have advocated a discovery-learning pedagogy, it is felt that a structured presentation of concepts and activities may produce a more rewarding development of programming skills. In addition, the report of the research on children learning to do LOGO programming conducted in the Bank Street College of Education (Pea, 1983) seems to confirm the need for instructional guidance. In this section, therefore, a sequence of LOGO concepts and activities are suggested in a series of units, each consisting of brief objective statements, activity instructions, program examples and notes. As some units comprise a fairly large number of tasks to perform, there is no specification as to the amount of time that should be devoted to each unit. Teachers work at their own pace. The language points in the demonstration programs have been arbitrarily selected; teachers are greatly urged to choose linguistic points of their interests and teaching needs. There is, at this stage, no research back-up to the sequencing and selection of the concepts and activities, which are purely based on careful consideration of the issues raised in Section II, and the writer's personal experience of learning LOGO. Feedback to the writer from interested teachers who have used the units will probably lead to the germination of meaningful research in the area.

It is assumed that teachers have no difficulty in the access to LOGO hardware and software. Any of the existing LOGO versions are just as good, but the examples given have been written in Terrapin LOGO. Reading materials recommended and utility procedures suggested are all based on the Terrapin version.

Unit One

Objective

Acquire a general understanding of LOGO operation and basic drawing commands through reading explanations and hands-on practice.

Activity

(1) Read the chapters listed below in Harold Abelson's APPLE LOGO or LOGO for the APPLE II (Byte - McGraw Hill, 1982):

1.A First Look at LOGO

1.1.Before Using LOGO

1.2.Using LOGO Commands

1.3.Introduction to Procedures

1.5.Modes of Using the Screen

4.Workspace, Filing, and Debugging

4.1.Managing Workspace

4.2.The LOGO File System

(2) Practise loading the LOGO language disk, and initializing own data disks.

(3) Practise drawing squares, rectangles and circles in the draw mode.

(4) Practise writing procedure definitions for the drawing of circles and blocks in nodraw mode, using commands such as FORWARD, BACKWARD, RIGHT, LEFT, PENUP AND PENDOWN.

(5) Practise using the simple editing features introduced in Chapter 1.2.2.

(6) Practise saving, erasing, and reading procedures on disk.

(7) Practise saving and reading pictures on disk.

Note

(A) To define a procedure is to give a set of instructions to the computer for a very specific purpose; to execute or run the procedure is to see what is produced as a result of the computer's response to the set of instructions given.

(B) When procedures are saved under a file name, all procedures in the workspace are saved under that name. When the file name is read, all procedures in it will be read.

(C) When the definition of a procedure is changed, but the name of the procedure is kept the same, the previous definition will be automatically erased.

(D) Changing the procedure name to another one, but keeping the same definition, produces two procedures of the same content or instructions.

Unit Two

Objective

Acquire a general understanding of basic list processing commands through reading of explantions and examples.

Activity

(1).Read the chapters listed below in LOGO for the APPLE II:

5.Numbers, Words, and Lists

5.3.Words

5.4.Lists

5.5.Naming

2.Programming with Procedures

2.2.Repetition and Recursion

2.2.2.Conditional commands and STOP

5.Numbers, Words, and Lists

5.6.Conditional Expressions and Predicates

5.7.Details on LOGO Syntax

(2) Read Text Box 8 -- The Magic of Recursion (P. 156 - 157) from the article listed below: Watt, D. LOGO : What makes it exciting? Popular Computing, August 1983.

(3) From what is explained in the chapters from LOGO for the APPLE II make a list on paper the functions of these commands, and the punctuation marks, such as [ ], ( ), " and :, that are associated with them: PRINT, WORD, SENTENCE, REQUEST, REPEAT, MAKE, IF...THEN...STOP, IF...THEN...ELSE..., IF ANYOF/ALLOF.

(4) Write on paper the function of using the procedure name in the same

procedure, as in the Boast example in LOGO for the APPLE II (p. 86), and in the Hanoi Tower example in "LOGO: What makes it exciting?". In each example, how is recursion terminated?

Note

(A) It is sometimes not easy to specify in words the functions of the commands. Examples and diagrams can be used for illustration.

(B) Recursion is a difficult concept; beginners are not expected to come to full grip with the idea, but it is useful to be aware of how it is used in specific examples. The Boast example shows one very elementary way of using recursion. In this example recursion is not terminated by means of any instruction in the program. It can only be stopped by pressing the CONTROL key and the G key.

Unit Three

Objective

Write programs using the commands PRINT, SENTENCE and REQUEST to produce exercises on completion drills, and develop programs that call sub-procedures, indicating the programmer's consciousness of modular structuring.

Activity

(1) Read the program listed below, and write similar ones on individual choice of linguistic learning points.

```
TO COMPLETION.1

PRINT [WHAT IS YOUR NAME?]

PRINT SENTENCE REQUEST [HOW DO YOU DO.  PLEASE COMPLETE THE
FOLLOWING SENTENCE BY USING THE GIVEN VERBS CORRECTLY.]

PRINT [YESTERDAY, SUPERMAN _____ MY BICYCLE.  (FLY BACK, REPAIR)]

PRINT (SENTENCE [YESTERDAY, SUPERMAN] REQUEST [MY BICYCLE.])

PRINT [IF YOU'VE TYPED 'FLEW BACK TO REPAIR', YOU'VE GOT IT RIGHT.]

END
```

(2) Expand the following program by adding other procedures similar to procedures EX.I and EX.2 used in the program.

```
TO COMPLETION.2

PRINT [WHAT IS YOUR NAME?]

PRINT SENTENCE REQUEST [HOW DO YOU DO.  PLEASE COMPLETE THE
FOLLOWING SENTENCES BY USING THE VERBS GIVEN IN BRACKETS.]

EX.1

EX.2
END
```

```
TO EX.1

PRINT [THE KOREAN PLANE _____ IN RUSSIAN TERRITORIES.  (SHOOT)]

PRINT (SENTENCE [THE KOREAN PLANE] REQUEST [IN RUSSIAN
TERRITORIES.])

PRINT ['WAS SHOT' OR 'HAS BEEN SHOT' IS THE CORRECT ANSWER.]

END


TO EX.2

PRINT [MANY COUNTRIES NOW __ RUSSIAN FLIGHTS.  (DECIDE, BOYCOTT)]

PRINT (SENTENCE [MANY COUNTRIES] REQUEST [RUSSIAN FLIGHTS.])

PRINT ['HAVE DECIDED TO BOYCOTT' IS THE ANSWER.]

END
```

Note

(A) When more than two items have to be combined by means of SENTENCE, parentheses are required.

(B) The program COMPLETION.2 can virtually grow indefinitely, but before expanding it, it may be useful to start checking the storage capacity available, by typing .NODES . At this command, the computer will give out a figure showing the number of nodes, indicating the memory space that is still available on that particular working disk.

Unit Four

Objective

Write programs using the commands REPEAT and MAKE, and the utility procedure SHOWPICTURE to develop short picture description exercises.

Activity

(1) Copy the utility procedure SHOWPICTURE, and use it as if it is a LOGO primitive or built-in command.

```
TO SHOWPICTURE :PICTURE.NAME
FULLSCREEN
READPICT :PICTURE.NAME
SPLITSCREEN
PRINT [PRESS ANY KEY TO CONTINUE]
MAKE "KEYPRESSED READCHARACTER
END
```

(2) Write programs to produce a simple diagram, and give instructions to describe it, e.g.

```
TO SQUARES
REPEAT 4[FD 50 RT 90] LT 90
REPEAT 2[FD 30 RT 90] FD 30
HIDETURTE
END
```

(After SQUARES has been executed, and the picture is on the screen, type SAVE "SQUARES to save the picture under the picture name Squares.)

```
TO DESCRIBE.PICTURE
SHOWPICTURE "SQUARES
```

```
PRINT [DESCRIBE THE POSITION OF THE SMALL SQUARE IN RELATION TO THE
BIG ONE.]

MAKE "DESCRIPTION REQUEST

PRINT [HAVE YOU USED EXPRESSIONS LIKE 'IS ATTACHED ' AND 'TO
THE RIGHT OF'.  PLEASE DESCRIBE THE POSITION ONCE MORE.  ASK YOUR
TEACHER TO READ IT.]

MAKE "DESCRIPTION REQUEST

END
```

Note

(A) When a picture is shown on the screen, there are only three lines available for text display and input; thus only short instructions and descriptions are possible.

(B) It is clear to use full and meaningful words as procedure and variable names, but there is the disadvantage that long words use up more computer memory space.

(C) In line, MAKE "DESCRIPTION REQUEST, in procedure DESCRIBE.PICTURE, what the user of the program types in is stored as the value under the variable name DESCRIPTION.

## Unit Five

### Objective

Write programs using the utility procedures PICKRANDOM and READNUMBER, and the command OUTPUT in addition to those used before, in particular MAKE and REPEAT, to generate sentences or short messages.

### Activity

(1) Read the following program, and write similar ones on individual choice of linguistic learning points.

```
TO EXAMPLE

MAKE "SUBJECT [THE PASSENGERS ON BOARD THE PLANE]

MAKE "PREDICATE [ALL DIED]

MAKE "TIME [LAST WEEK]

MAKE "PERIOD [.]

MAKE "EXAMPLE (SENTENCE :SUBJECT :PREDICATE :TIME :PERIOD)

PRINT [READ THE FOLLOWING SENTENCE, AND TYPE OUT ITS PRINCIPAL
NOUN.]

PRINT :EXAMPLE

MAKE "ANSWER REQUEST

PRINT [WHAT HAVE YOU TYPED?  THE ANSWER IS 'PASSENGERS'.]

PRINT [CAN YOU EXPAND THE SENTENCE BY SAYING WHERE THE PASSENGERS
DIED?  TYPE THE WHOLE SENTENCE.]

MAKE "ANSWER REQUEST

PRINT [PLEASE ASK YOUR FRIEND TO CHECK YOUR SENTENCE -- JUST IN
CASE YOU'VE MADE A CARELESS SPELLING MISTAKE.]

END
```

(2) Copy the utility procedures PICKRANDOM from p. 103 and READNUMBER from p. 102 in LOGO for the APPLE II, and develop programs similar to

the one below:

```
TO MESSAGES

PRINT [HOW MANY MESSAGES WOULD YOU LIKE TO HAVE?]

MAKE "TIMES READNUMBER

REPEAT :TIMES [MESSAGE]

END


TO MESSAGE

PRINT [SEND A MESSAGE TO ONE OF YOUR FRIENDS.  TYPE IT OUT
CLEARLY.]

MAKE "MESSAGE REQUEST

PRINT [NOW SIGN OFF BY TYPING YOUR OWN NAME.]

MAKE "OWN.NAME REQUEST

PRINT [HERE IS THE COMPLETE MESSAGE.]

PRINT SENTENCE [DEAR] NAME

PRINT :MESSAGE

PRINT (SENTENCE CLOSINGS [ - ] :OWN.NAME)

PRINT [ASK YOUR FRIEND TO READ YOUR MESSAGE.  SEE IF HE/SHE
UNDERSTANDS IT, AND THINKS IT HAS BEEN WELL WRITTEN.]

END


TO NAME

MAKE "NAME [BOB SUE ALICE DANNY]

MAKE "NAME PICKRANDOM :NAME

OUTPUT :NAME

END
```

```
TO CLOSINGS
MAKE "CLOSINGS [LOVE [BEST REGARDS] [SEE YOU]]
MAKE "CLOSINGS PICKRANDOM :CLOSINGS
OUTPUT :CLOSINGS
END
```

Note

(A) The MESSAGES program is modelled on POSTCARDS in "A beginner's guide to LOGO" (Abelson, 1982a).

(B) The procedure name and the variable name can be the same in LOGO.

(C) When the value of a variable is called, a colon : must be used before the variable name, but when the result of a procedure is used, no colon is needed.

(D) If an item in a list consists of more than one word, square brackets must be used, e.g. [LOVE [BEST REGARDS] [SEE YOU]].

(E) MAKE can be used to store the value of a variable and also to change the value. For example, in MAKE "NAME [BOB SUE ALICE DANNY], the value of NAME consists of four names, but in MAKE "NAME PICKRANDOM :NAME, the value of NAME has become only one of the names, randomly picked from the list of four.

Unit Six

Objective

Write programs for vocabulary or reading comprehension exercises, using the commands GO and IF ...THEN ...STOP and ELSE. A counter device, used to change the conditions after they have taken place for a number of times, is also introduced.

Activity

(1) Note the use of IF...THEN...STOP and ELSE in the following program:

```
TO DEFINITION.1

PRINT [WHAT DO YOU CALL THE SHIP WHICH CAN SAIL UNDER WATER?]

MAKE "REPLY REQUEST

IF :REPLY = [SUBMARINE] THEN PRINT [GOOD] STOP
ELSE PRINT [SORRY, THE ANSWER IS 'SUBMARINE'.]

END
```

(2) Analyse the program below by drawing diagrams to indicate the flow of control, or the change in conditions.

```
TO DEFINITION.2

MAKE "COUNT 1

LOOP:  PRINT [WHAT DO WE CALL THE SHIP WHICH CAN SAIL UNDER WATER?]

MAKE "REPLY REQUEST

IF :REPLY = [SUBMARINE] THEN PRINT [GOOD] STOP

MAKE "COUNT :COUNT + 1

IF :COUNT < 3 PRINT [SORRY, TRY AGAIN.] GO "LOOP

PRINT [THE WORD CONSISTS OF A PREFIX WHICH MEANS 'UNDER' AND A
ROOT THAT SUGGESTS 'THE SEA'.  WHAT IS IT?]

MAKE "REPLY REQUEST

IF :REPLY = [SUBMARINE] THEN PRINT [YOU'VE GOT IT.] STOP
```

```
PRINT [THE ANSWER IS SUBMARINE.]
END
```

(3) Write procedures similar to DEFINITION.2 as sub-procedures to be called by a top procedure. These exercises can be on vocabulary or on a piece of text read.

Note

(A) GO takes a word used as a label as an input, and transfers execution to the line with that label in the same procedure; in the DEFINITION.2 example, the label word is LOOP. Labels are defined by typing them at the beginning of the indicated line followed by a colon.

(B) In the line, IF :COUNT < 3 PRINT [SORRY, TRY AGAIN.] GO "LOOP, the meaning is if the value of COUNT is less than 3, execute the PRINT statement, and then the instructions starting from LOOP: again; however, if the value of COUNT is equal to or greater than three, the instruction following this line is executed, which is PRINT [THE WORD CONSISTS...].

(C) COUNT in the given procedure DEFINITION.2 is used to control when further help to the user, in this case the meaning of the prefix and root of the word, is to be provided.

Unit Seven

Objective

Write programs using conditional commands to produce exercises on grammar or language usage. In addition to IF...THEN...STOP... ELSE, ANYOF/ALLOF is introduced. Have consolidation work on the modular structure of LOGO programming.

Activity

(1) Read the program below used to help users practise with writing complete sentences instead of just long fragments. Modify the program, by decomposing it into modules, and write instructions (definition) for the sub-procedure, CHECK2.

```
TO MANY.EXAMPLES

MAKE "WHO [[SUPERMAN] [MY FATHER] [LITTLE JOHNNY] [THE WAR HERO]
[HER BOYFRIEND] [ ]]

MAKE "ACTION [[DANCED MADLY] [RESCUED THE PRINCESS] [THREW THE CAKE
AT THE PRIME MINISTER'S FACE] [GOT LOST] [STARTED PLAYING WITH
LOGO] [ ]]

MAKE "WHERE [[IN THE WHITE HOUSE] [IN THE DARK FOREST] [AT SHERITON
CENTRE] [ON BOARD APPOLO 10] [IN THE PARTY]]

MAKE "WHEN [[YESTERDAY,] [LAST SPRING,] [ON EASTER SUNDAY,] [TWO
MONTHS AGO,] [AT SIX THIS MORNING,] ]

MAKE "PERIOD [[.] [ ]]

MAKE "WHO PICKRANDOM :WHO

MAKE "ACTION PICKRANDOM :ACTION

MAKE "WHERE PICKRANDOM :WHERE

MAKE "WHEN PICKRANDOM :WHEN

MAKE "PERIOD PICKRANDOM :PERIOD

MAKE "EXAMPLE (SENTENCE :WHEN :WHO :ACTION :WHERE :PERIOD)
```

```
PRINT :EXAMPLE

PRINT [IS THIS A FRAGMENT OR A COMPLETE SENTENCE?  TYPE F FOR
FRAGMENT AND S FOR SENTENCE.]

MAKE "REPLY REQUEST

IF :REQUEST = [F] THEN CHECK1 ELSE CHECK2

END


TO CHECK1

IF ANYOF (:WHO = [ ]) (:ACTION = [ ]) PRINT [YOU'RE CORRECT.
DEVELOP IT INTO A SENTENCE PLEASE.  ASK YOUR GROUP LEADER OR
TEACHER TO CHECK IT.] MAKE "ANSWER REQUEST
ELSE PRINT [NO, IT'S A COMPLETE SENTENCE.  READ IT ONCE MORE.]
CHECK.PERIOD

END


TO CHECK2

......


TO CHECK.PERIOD

PRINT [IS THERE A PERIOD AT THE END OF THE SENTENCE?  PLEASE TYPE
YES OR NO.] MAKE "ANSWER REQUEST

IF :REPLY = [YES] THEN CHECK
ELSE PRINT [TYPE THE SENTENCE ONCE MORE.  PUT IN THE MISSING
PERIOD.] MAKE "ANSWER REQUEST

END


TO CHECK

MAKE "LASTWORD LAST :EXAMPLE

IF :LASTWORD = ".  THEN PRINT [GOOD]
ELSE PRINT [SOMETHING IS MISSING.  TYPE THE SENTENCE ONCE MORE.
PUT IN THE MISSING PART.] MAKE "ANSWER REQUEST

END
```

(2) Write programs similar to MANY.EXAMPLES on individual choice of linguistic learning points, trying to use sub-procedures for the different parts or modules of the program. Include the REPEAT command so that the user can decide the number of examples or exercises to be generated.

Note

(A) In the given sub-procedure CHECK1, the different conditions specified after ANYOF have to be in parentheses. The IF ANYOF ... ELSE test operation means that if any of the conditions specified is true, carry out the commands that follow the conditions immediately; otherwise, perform the commands given after ELSE. Several commands can be given after IF ANYOF and ELSE, but as they have to be in the same line, heed must be paid to the maximum number of characters (256) that can be hit on one line before pressing the RETURN key.

(B) In the sub-procedure CHECK, the command LAST has been used. The use of this and related commands will be practised in the following unit. But care must be given to the " instead [ ] used before the period in the line, IF :LASTWORD = ". THEN PRINT [GOOD]. The last item of :EXAMPLE (which is a sentence) is a word, not a list any more; therefore the quotes for words have to be employed.

(C) For the first activity, MANY.EXAMPLES can be decomposed in this way:

```
TO MANY.EXAMPLES (Top-procedure)
MAKE "EXAMPLE (SENTENCE WHEN WHO ACTION WHERE PERIOD)
PRINT :EXAMPLE
CHECK.SENTENCE
END
```

```
TO WHEN

MAKE "WHEN [[YESTERDAY,] [...] [...]]

MAKE "WHEN PICKRANDOM :WHEN

OUTPUT :WHEN

END


TO WHO


TO ACTION


TO WHERE


TO PERIOD


TO CHECK.SENTENCE


TO CHECK


TO CHECK1


TO CHECK2

IF ANYOF (:WHO = [ ]) (:ACTION = [ ]) PRINT [SORRY.  IT'S A
FRAGMENT.  DEVELOP IT INTO A SENTENCE AND ASK YOUR GROUP
LEADER OR TEACHER TO CHECK IT.] MAKE "ANSWER REQUEST
ELSE PRINT [CORRECT.  IT'S SENTENCE.] CHECK.PERIOD

END


TO CHECK.PERIOD
```

## Unit Eight

### Objective

Write programs using recursion and the commands FIRST, LAST, BUTFIRST and BUTLAST in addition to those used in previous units to produce exercises that can be used to look for key words in students' writing.

### Activity

(1) Read the following program that enables the user to develop a sentence into a short paragraph, and be aware of the occurrence of the bridging word 'and' in his/her writing. Analyse the program and draw diagrams to indicate the steps involved in checking whether the word 'and ' has been used in a sentence. Identify the devices used to keep count of the number of times the word has been used, and to mark the beginning of a new sentence. Explain why recursion, the calling of STYLE1 in the procedure STYLE1, has to be used in this program, and how it is terminated.

```
TO STORY
PRINT [USE THE FOLLOWING AS A TOPIC SENTENCE, AND DEVELOP IT INTO A
SHORT PARAGRAPH OF THREE SENTENCES.]
SENTENCE1
SENTENCE2
SENTENCE3
MAKE "PARA (SENTENCE :SENTENCE1 :SENTENCE2 :SENTENCE3)
PRINT :PARA
PRINT [ ]
STYLE
END
```

```
TO SENTENCE1

MAKE "SENTENCE1 [LAST CHRISTMAS, MY FATHER FLEW THE CAKE AT THE
PRIME MINISTER'S FACE IN THE PARTY.]

PRINT :SENTENCE1

PRINT [ ]

END


TO SENTENCE2

PRINT [NOW WRITE THE NEXT SENTENCE.  DON'T FORGET TO PUT A PERIOD
AT THE END OF THE SENTENCE.]

MAKE "SENTENCE2 REQUEST

PRINT [ ]

END


TO SENTENCE3

PRINT [WRITE THE LAST SENTENCE PLEASE.]

MAKE "SENTENCE3 REQUEST

PRINT [ ]

END


TO STYLE

MAKE "CHECKLIST :PARA

MAKE "CHECKWORD FIRST :CHECKLIST

MAKE "N 0

MAKE "M 1

STYLE1

END
```

```
TO STYLE1
IF :CHECKWORD = "AND MAKE "N :N + 1
STYLE2
IF :CHECKLIST = [ ] STOP
STYLE1
END

TO STYLE2
MAKE "END (LAST :CHECKWORD)
IF :END = ".  THEN PRINT (SENTENCE [YOU HAVE USED] :N [AND/S IN
SENTENCE] :M [.]) MAKE "N 0 MAKE "M :M + 1
MAKE "CHECKLIST (BUTFIRST :CHECKLIST)
IF :CHECKLIST = [ ] STOP
MAKE "CHECKWORD (FIRST :CHECKLIST)
END
```

(Suppose for SENTENCE2 and SENTENCE3, the user has typed in these sentences: The prime minister was very angry, and we could see that even his neck had turned red. He fumbled for a paper towel to clean up the mess on his face, and he looked mad, and he clenched his fist, and in his rage, his lips quivered into a twisted O. The program will give out these messages:

YOU HAVE USED 0 AND/S IN SENTENCE1

YOU HAVE USED 1 AND/S IN SENTENCE2

YOU HAVE USED 3 AND/S IN SENTENCE3)

(2) As a final project in this first LOGO programming adventure, produce a program that can be used to check the occurrence of certain key words in a piece of writing, or in an answer to a question.

Note

(A) When reading the programs given in this unit, care must be taken to differentiate the use of the word sentence in its everyday meaning, and sentence as the LOGO operator.

(B) In the given procedures the line, PRINT [ ], is simply used to produce an empty line.

(C) In the program, :PARA is the whole paragraph; :CHECKLIST is first made exactly the same in value as :PARA, but as the program is executed, the first word in the paragraph is removed one by one by means of the command BUTFIRST after being checked if it is the word 'and'. The program stops, when all the words in the paragraph are removed. (This is indicated by an empty list [ ].) :CHECKWORD is the first word in the paragraph or what is remained in the paragraph, as the words are being compared and taken away. Since it is a word, the value with which it is compared has also to be a word. Thus, in line, IF :CHECKWORD = "AND MAKE "N :N + 1, in procedure STYLE1, it will be wrong if AND is treated as a list. In other words, if the line is written as IF :CHECKWORD = [AND] MAKE "N :N + 1, the program won't work.

Conclusion

As the topic of the paper points out, the discussions and suggestions are on introductory LOGO programming. While it is believed that teachers will not feel bewildered at some of the slightly more complicated programming concepts and operations, it is also hoped that teachers will not be contented with the limited activities proposed. There are certainly more advanced list processing features in LOGO, but as research reports have revealed, even experienced programmers use only a small subset of language constructs frequently. Hence, learning more LOGO commands does not seem to be the urgent task of teachers who would like to go further in LOGO programming. Rather the experimentation of the commands and features introduced in more varied and interesting language arts applications is of greater significance, for only then can some powerful computer ideas be more fully grasped, and programming be made a means to better language instruction and learning.

References

Abelson, H. (1982a). A beginner's guide to LOGO: LOGO is not just for kids. Byte, 7(8), 88-115.

Abelson, H. (1982). LOGO for the APPLE II, Byte - McGraw Hill.

Austin, H. (1976). Teaching teachers LOGO: the Lesley experiments. Massachusetts Institute of Technology, LOGO Memo 23.

Burns, M. (1980). Getting kids ready for computer thinking - thoughts for teachers, Grades 4-8. The Computing Teacher, 8(1), 28-32.

Du Boulay, B. & O'Shea, T. (1981). Teaching novices programming. In Coombs, M.J. & Alty, J.C. Computing Skills & the User Interface, New York: Academic Press, 147-200.

Harvey, B. (1982). Why LOGO? Byte, 7(8), 163-195.

Marty, F. (1981). Reflections on the use of computers in second language acquisition - I. Systems, 9(2), 85-98.

OISE Survey & Analysis of Microcomputer Software for Language Arts. Interim Report: Year 1, May, 1983.

Papert, S. (1980). Mindstorms: Children, Computers & Powerful Ideas, New York: Basic Books.

Pea, R.D. (1983). LOGO programming and problem solving. Chameleon in the classroom: developing roles for computers (Technical Report No.2), Bank Street College of Education, 25-33.

Pea, R.D. & Kurland, D.M. (1984). On the cognitive effects of learning computer programming. New Ideas in Psychology, 2(2), 137-168.

Statz, J. (1973). Syracuse University LOGO project: final report. Syracuse University, New York.

The National LOGO Exchange, 3(8), April, 1985.

Watt, D. (1983a). Learning with LOGO, McGraw-Hill Co.

Watt, D. (1983). LOGO: what makes it exciting? Popular Computing, 2(10), 106-166.